Enterprise Programming Assignment by Viran Gohil, StuID-15500582

The following report will highlight software engineering techniques used, where they were used and the advantages and disadvantages of adopting a specific technique.

The primary pattern used was the Model, View and Controller (MVC). In the project name 'Try 5', the servlets are the controller. The servlets receives requests, retrieves data from the DAO and sends data to the appropriate viewer. The DAO is the model. The DAO outlines the methods for interacting with storage. The viewer receives the object and converts. Within this project, the course information can be viewed by invoking the jsp files in the results section. The controller also sends course information to the Ajax enable user interface.

The main benefit of the MVC pattern is its modularity. The separation enables an individual to develop without retaining everything in the memory or having to write separate notes to the code. In an enterprise environment, the pattern enables teams of programmers to work on the same applications without continuous overlap. Furthermore, this methodology enables the reuse of code in other applications. For example, the CourseDBConnect class in the 'try 5' project has the sole purpose of offering a public a method which can be invoked to give a single instance of the database connection. With the simple change of database url, username and password, the class can be re-used in any similar project. As an aside, the CourseDBConnect class contains a singleton pattern in that is restricts the instantiation of a class to one object. In this project, this instance has a connection method.

Fig 1 Demonstrating Modularity.

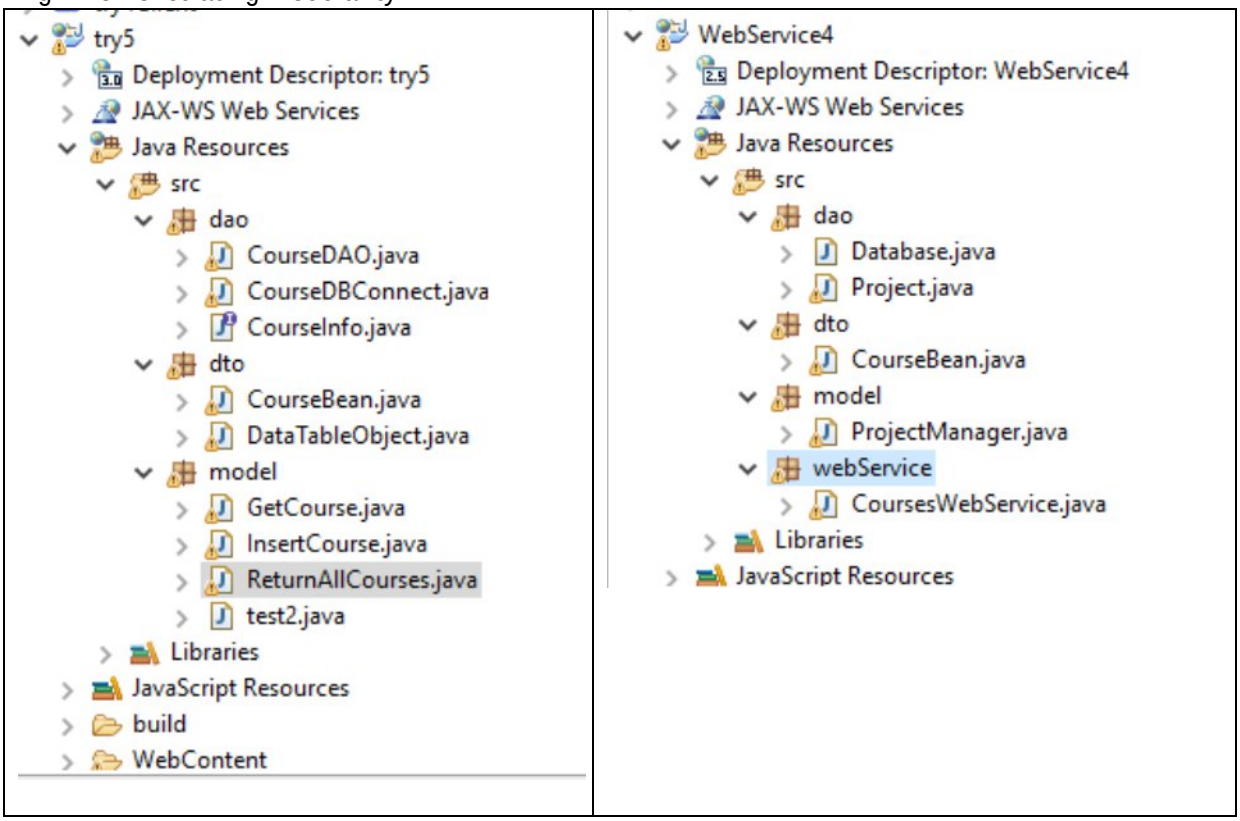

Fig 2 Singleton Pattern

```
public class CourseDBConnect {
    private Connection con;

    /**
     * Singleton Pattern private constructor so cannot get instance outside the
     * class.
     *
     */
    private static CourseDBConnect CONNECT = new CourseDBConnect();

    private CourseDBConnect() {
    }

    /**
     * getConnect is the method to give public access to an instance of the
     * class.
     *
     */
    public static CourseDBConnect getConnect() {
        return CONNECT;
    }

    /**
     *
     * Method to get instance of DB connection
     *
     */
    public Connection getLink() {
```

Furthermore, MVC is of benefit as it allows one to change the user interface as a separate module without disturbing the code separating the UI designer and the faster pace of UI design change from the business logic programming. This separation also means that the UI can be developed to return data in unlimited ways.

Notwithstanding the clear benefits in an enterprise environment, the MVC pattern is unsuitable for smaller applications due to the increased intricacy in development. The key aspects of this are the replication of code in modules and requirement for separate modules to interact and potential time lost correcting errors emanating from the potentially adverse effects of code in one module effecting code in other modules and vice versa.

A second pattern we are using is the Data Access Object Pattern. The CourseInfo class is the DAO interface whilst CourseDBConnect and CourseDAO are the Implementation classes. The advantages of the pattern is its role as a module or layer that is only concerned with interaction with the data store. Separation means that changes in the data store do not affect the application logic.

The third pattern we have utilised is the Data Transfer Object. This is a simple method of holding multiple attributes in a single object. The benefit of this is that it reduces both the number of method calls and amount of system resources that would be utilised transferring lone attributes.

Fig 3 Shows how user four different form input variables are saved as type then inserted into a single CourseBean object.

```
public void init() throws ServletException {
    this.insertCourseObj = new CourseDAO();
    this.course = new CourseBean();
}

protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

    //Get course information from user. Must be converted to String.
    String cName = request.getParameter("CourseName");
    String cCredits = request.getParameter("CourseCredits");
    String cDuration = request.getParameter("CourseDuration");
    String cTutor = request.getParameter("CourseTutor");


    //Set course information into a CourseBean object.
    this.course.setCourseName(cName);
    this.course.setCourseCredits(cCredits);
    this.course.setCourseDuration(cDuration);
    this.course.setCourseTutor(cTutor);
    System.out.println("GOT >>> " + this.course.getCourseName());

    //Pass the CourseBean object into the CourseDAO object applying the addcourse to
    //databse methid.
    if (this.insertCourseObj.addCourse(this.course)) {
        System.out.println(" 1 = courses saves successfully");
    } else {
        System.out.println(" 0 = failure");
```

Regards refactoring, the only examples are the CourseDBConnect class and my-demo-table script with code to connect and return the populated JQuery data table in the browser. The benefits of the first have already been discussed. For the second, it would have required writing multiple ajax functions to achieve the same as one simple function written within the JQuery data table plugin API. Thus, the JQuery libraries key benefit is that it is much more simpler than writing AJAX functions. The only minimal negative of utilising it is that it requires the javascript library to be loaded so is a resource drain in that regard. It must be emphasised that this is an insignificant negative compared to the advantages.

Fig 4 JQuery DataTable Plugin script for creating the table in Fig 5.

```
$(document).ready(function() {

    $(".jqueryDataTable").dataTable({
        "bProcessing": false,
        "bServerSide": false,
        "sAjaxSource": "./returnAllcourses",
        "bJQueryUI": true,
        "aoColumns": [
            { "mData": "courseName"},
            {"mData": "courseCredits"},
            {"mData": "courseDuration"},
            {"mData": "courseTutor"}

            ]
    });
});
```

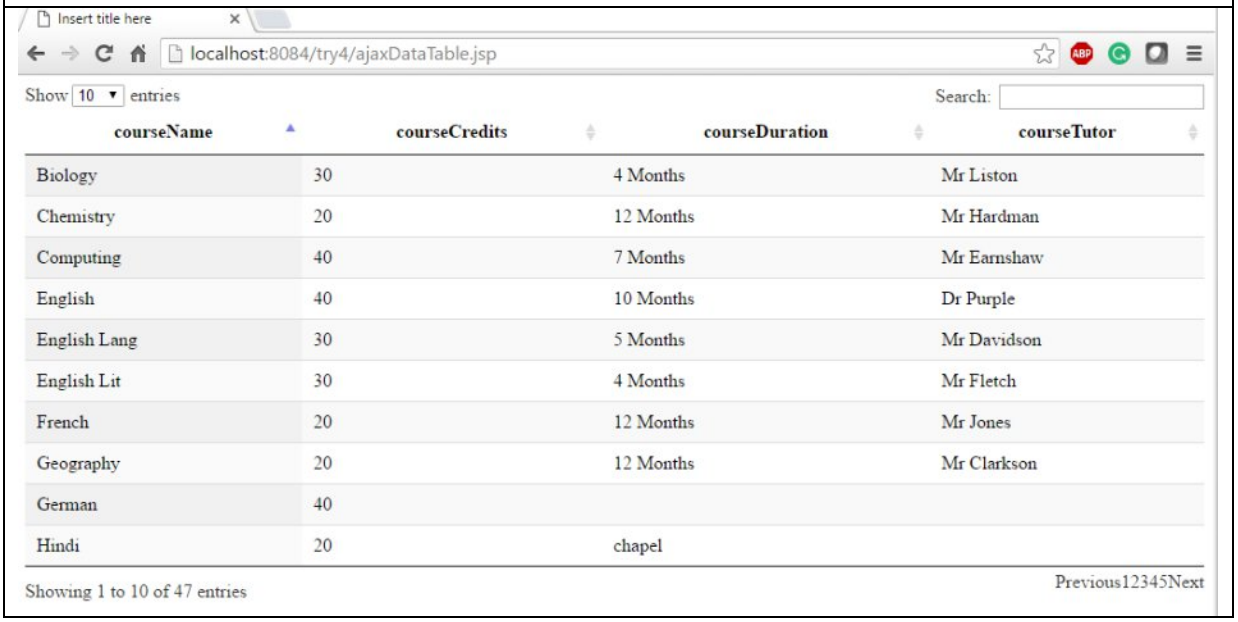

Fig 5 JQuery DataTable. The facility to search all columns in the search box top right makes the 'WHERE' clause servlet in project 'try5' obsolete in AJAX.

Additionally, on this note of API's, we used the GSON library to convert a Java ArrayList to a JSON object to output as JSON. We use Google's Gson library to implement this. The alternative to GSON would have been a 'for each' loop written in the jsp document with. Gson significantly reduces the complexity achieving this in a few lines of syntax whilst the 'for each' loop requires greater concentration and more research in ensuring one has imported the right jar files.

Fig 6 Comparison code and explanations. GSON library versus a 'for each' loop in jsp.

Changing an ArrayList to XML in JSP. Similar amount of coding required for JSON.	<?xml version="1.0" encoding="ISO-8859-1"?> <%@ page contentType="text/xml" %> <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %> <ReturnAllCoursesServlet> <c:forEach var="item" items="${sList}" > <CourseBean> <courseName>${item.courseName}</courseName> <courseCredits>${item.courseCredits}</courseCredits> <courseDuration>${item.courseDuration}</courseDuration> <courseTutor>${item.courseTutor}</courseTutor> </CourseBean> </c:forEach> </ReturnAllCoursesServlet>
Alternatively, use this legible, simpler code in the Servlet. Only write ${Json} in the JSP page.	Gson gson1 = new GsonBuilder().setPrettyPrinting().create(); String json = gson1.toJson(javaArrayORobject); outputPage = "/WEB-INF/results/course-json.jsp"; out.print(json);

Unfortunately, we have run out of time to discuss the code in terms of layout, readability and quality. Furthermore, we have been unable to comment on the best API (SOAP, REST or http).